# REAL THROUGHPUT MEASUREMENTS COMPARISON BETWEEN UNOPTIMIZED AND OPTIMIZED NETWORK CARDS

**Okta Nurika[1,*], Mohd Fadzil Hassan[1,*], Nordin Zakaria[1,*]**
[1]*Department of Computer & Information Sciences, Universiti Teknologi PETRONAS, 32610 Perak, Malaysia*
[*]For correspondence; E-mail: okta.rider@gmail.com

***ABSTRACT:*** *This paper is an expansion of our previous publication about simultaneous multiple network cards optimization in a data centre simulation, using genetic algorithm (GA). The generated optimal solutions from our simulations, which represent optimal settings for multiple network cards need to be tested in the real hardware environment. Each optimal network card setting is meant for different data transmission characteristic. In a real network environment, we streamed every optimal network card setting with their respective data transmission and recorded the throughput rate, and then compared it against the throughput from the network card's default setting (unoptimized). The comparison results infer that all of our optimal network card settings provide higher throughput than the default setting. Furthermore, our optimal settings exhibit capability to prevent the CPU from crashing, as a result from being overwhelmed by the stream of packets. This effect comes from the optimal settings' ability to minimize kernel interrupt generation, hence the CPU cycles can be preserved. The preservation of CPU cycles leads to lower heat generation and lower power consumption, which is economically beneficial for the data centre itself.*

**Keywords:** Network card, active wait, passive wait, watermark mode, optimization, throughput, CPU cycles, genetic algorithm.

## 1. INTRODUCTION

Optimization is an important research area, where the existing system is adjusted to improve its performance. A system typically contains some adjustable combinatorial parameters, which when optimized will enhance the system's performance, compared to the default setting.

In our case, the optimization area is about network card optimization, which is a part of network optimization and hardware optimization. Our simulations of network card optimizations have been conducted and published in our previous paper [1]. However, its real environment's validations of throughput between the unoptimized and optimized network card setting have not been published, thus this paper will fulfill that research gap by presenting the mentioned comparisons.

The existing network optimizations that we have reviewed are listed in [2] and [3]. Other worth mentioning papers about measurement between unoptimized and optimized system are for example [4], which compares the applications performances between unoptimized and optimized compiler, [5] that compares between unoptimized and optimized android smartphone on long survey, benchmarking between unoptimized object-oriented codes and the optimized ones by [6], comparison of performances between unoptimized and optimized routing in packet switched networks by [7], comparison between unoptimized and optimized video codecs' performances by [8], benchmarking between the unoptimized and optimized FPGA (Field-Programmable Gate Array) embedded processors by [9, 10] that measures the performance differences between optimized incidence tree parametrized systems and the unoptimized ones, and other works that are too much to mention all in this paper.

Our research results presentations in this paper are an important visualization of the degree of improvements that can be achieved by optimizing network card parameters. The network card throughput improvements will be seen for different transmission sizes with different packet sizes. Furthermore, besides increasing throughput, the optimal adjustment of network card will also reduce the usage of CPU cycles, which in turn will preserve the CPU cycles for other processes. This optimization's after-effect relates to [11], which argues that optimal CPU usage prevents overheating and unexpected shutdown. Moreover, additional benefits from CPU cycles preservation include lower heat generation and lower power consumption, which subsequently will prolong the CPU lifetime. As informed by [12] that the reduction of heat will also lower the resistance, which the electrons have to pass through to switch states, therefore lower heat will improve the CPU longevity. Automatically, the longer lifetime of CPU saves the cost of hardware replacement or upgrade. The avoidance of hardware replacement/upgrade eventually eliminates downtime of the data centre, caused by the replacement process. Therefore, the customers' satisfaction and profit can be sustained.

## 2. EXISTING NETWORK CARD OPTIMIZATIONS

This section explains about current network card optimization practices that we complement with our own methodology as conducted in [1].

Network card's kernel interrupt requests can be optimally allocated to a group of processors (CPUs) [13], to prevent certain CPU from being overwhelmed by too much interrupt requests. In other words, this network card optimization is directed to make processors more efficient. He also informs about NAPI packet polling within network card to reduce the number of interrupt requests, but does not specifically mention whether it is polling based on timer or amount of packets. However, both timer and amount of collected packets based polling optimization are parts of our research objectives.

Authors in [13], further converses about the offload feature for transfer of packet handling and checksum calculation task from the CPU to the network card. This is another method to make CPU more efficient, yet on the other hand it risks the performance of the network card. Another offload of CPU task to the network card is called TCP segmentation offload, which has another intention to improve CPU's efficiency,

while risking the network card in the process. Generic segmentation offload feature is also available, which can segment transmission from other than TCP protocol (generic). The next network card setting informed by [13] to reduce CPU load is called Large Receive Offload, where the network card will combine several Ethernet frames into one receive to save CPU cycle, but on the other hand it gives additional task to the network card.

Moreover, [13] also describes special featuresthe in Intel Dual Core and Intel Xeon Quad Core processor that can move network data more efficiently. However, other types of processor may not have this feature.

In conclusion, most of network card adjustments discussed by [13] are meant to preserve the CPU cycles for computation purposes instead of networking processes. The optimization inside the network card itself is seen as a research gap that we are about to fill up in this paper.

Distributing interrupt requests of multiple network adapters to different CPU cores is informed by [14]. This is again an act to preserve CPU cycles. A distinct feature of Mellanox network card as described by [14] is a low latency request to lower the power consumption.

The next method to preserve CPU cycles informed by [14] is the interrupt moderation tuning. However, there is no specific accurate guideline as it is based on loose estimation about transmission and receive rate, it is also without taking into account the certain transmission size and particular desired minimum throughput when receiving the transmission.

A risky method to allocate CPU to handling specific protocol process above the interrupt handler is Receive Packet Steering (RPS) ([14]). The risk involves its necessity to recompile the kernel, which may harm the system to not recognize the hardware at all.

Network card's timer based polling is also described by [14]. However, it does not explain about optimizing it to achieve specific throughput on receiving transmission and/or the minimization of interrupt generation based on specific transmission size. This optimization gap is complemented by this paper as it is one of our actual research objectives.

A case study reported by [15] discusses about improving FedEx data centre by upgrading their network cards to 10Gb Ethernet and their existing network cables to Direct Attach Twinax cabling, which only had 7 meters of length. This way, however, required the conversion of existing multiple physical links into virtual links using VLAN trunking. Furthermore, it did not instantly improve the network receive throughput to 10Gbps, instead the initial recorded throughput was only around 1/20 of the maximum 10Gbps. Another factor that affected the throughput was the choice of file transfer protocol, where the ones without cryptography improved throughput significantly. While in virtualized environment, the usage of multiple virtual machine (VM) instances gave higher throughput than the single one. All around, maximizing 10Gb network card needs some tool capability tests, system (BIOS) adjustment, VM configuration, and hardware configuration that often involves vendor consultation. Hence, hardware upgrade seems to trigger long installation and deployment time.

Modification of Intel network card kernel driver to moderate interrupt, according to packet rate instead of byte rate was done by [16]. Their algorithm was proven to be better than the Intel's default, however, it does not consider the embedded network card speed and the transmission size. Additionally, their algorithm does not consider throughput maximization, instead it focuses on interrupt and latency minimization. Furthermore, the implementation requires kernel change, and it is not widely deployed to all types of network card. Similar work is also accomplished by [17] using different algorithms.

Another interrupt moderation work is proposed by [18] who estimate the combination of interrupt and timer based polling. It is based on the principles that kernel interrupt is generated when a packet comes, while poll instruction is always executed even if there is no packet received. Thus, polling is wasting CPU cycles when there is no packet received. On the other hand, interrupt will excessively consume CPU cycles when there is acontinuous big transmission. This mixed approach brings side effect, where the continuous enablement/disablement of interrupt everytime polling is activated causes added CPU cycles. Such effect is discussed in [19] and [20]. The loose estimation of packet rate makes the interrupt-polling combination even more challenging, because there is no specification of transmission size. Furthermore, this method has afeasibility challenge to be implemented since it requires modification of network card driver. Additionally, another polling feature that uses watermark value was also not taken into their optimization case.

A similar method to [18] is [21], where they also combine interrupt and timer based polling that requires kernel recode, based on the loose estimated incoming packet rate. The objective of their optimization moderation is towards the minimization of the number of interrupts. It is emphasized by [21] that interrupt is expensively generated by high hardware and software events, which involve multiple processes as explained by [22]. Subsequently, they also argue that timer based polling is more appropriate for big data transmission.

The next work of combining interrupt and timer based polling was conducted by [23], who modified Linux kernel version 2.6.15 to preserve CPU cycle. Its optimization is based on packet rate estimation similar to the previously mentioned works. Therefore, it contains similar limitations, such as there is no accurate segmentation of transmission size, it is limited to acertain kernel version of specific network card, there is no specific range of values of the network card interrupt handling, and it does not include polling based on amount of packets captured.

The previously mentioned network card optimization works basically treat all packet transmissions by prediction, which adds extra computation cost and makes it more challenging to achieve the proper network card configuration. In our work, we specifically detail the transmission size as the optimization factor to make the network card configuration more accurate. The certain transmission size settings are to prioritize the important transmissions that flow inside a data centre. Added to it is the polling based on amount of packets received (watermark value), which makes our network card optimization contain more solution possibilities along with the inclusion of specified network card speed. Our method also does not require kernel coding, hence it is seen as a safer

and faster optimization way. The exclusion of kernel recoding also makes our network card optimization method compatible with multiple operating systems. Another extra feature by our method is the inclusion of both throughput maximization and kernel interrupt minimization [24], as our objectives. Finally, our method includes simultaneous multiple network cards optimization, because in real life, data centre contains multiple network devices with each consisting of multiple network cards. This is doable because of our adjustable simulation model [1].

## 3. PROPOSED RESEARCH

The network card parameters optimization that we have conducted revolve around 3 parameters, which are active wait mode, passive wait mode, and watermark mode. Active wait mode is when the network card generates kernel interrupt to process packets everytime a packet comes. Passive wait is a mode that uses timer based polling, before kernel interrupt is generated to indicate packets processing. Next, watermark mode is a polling based on amount of received packets (in Byte), before kernel interrupt is activated to process the received packets. The optimal choice of either one of these network card modes for a specific transmission, will increase the throughput and preserve the CPU cycles. More details about these network card modes can be read in our previous paper [1].

The simultaneous multiple network card optimizations reflect a data centre, with each network card handles different transmission characteristics. There are 47 different transmissions in total. After the genetic algorithm (GA) based optimizations were completed [1], the optimal network card solutions for all transmissions were tested in real physical network environment against each network card's default setting (unoptimized), to see if the optimal settings provided better throughputs than the default setting.

## 4. EXPERIMENTAL DETAILS

The data transmissions were generated using Linux based hping3 [25] software, while the network card modules were using PF_RING version 5.4.6 [26], which provided all the previously mentioned 3 network card modes. All the network card optimal settings' throughputs were compared with the ones from PF_RING's default setting. The throughput is calculated by dividing the amount of total packets (Bytes) acknowledged to hping3, over the duration (seconds) for hping3 to finish sending data transmission to the PF_RING network card module. The unit of throughput will be converted from Bytes/seconds to Megabit/seconds or Mbps. There are 47 total transmissions, which will be handled by 47 different optimal network card settings. These compiled comparisons against the default PF_RING setting are presented in the next section. Additionally, the respective complete list of optimal network card setting for every transmission can be read in our previous publication [24].

## 5. RESULTS AND DISCUSSION

From Figure 1 and Table 1 above, it is inferred that all the optimized versions of network cards always generate higher throughputs than the default setting (unoptimized), with the average throughput improvement of 4.6%. There is a notable case for Node 3, where the default setting caused the machine to crash as a result of its inability to handle the stream of packets transmission.

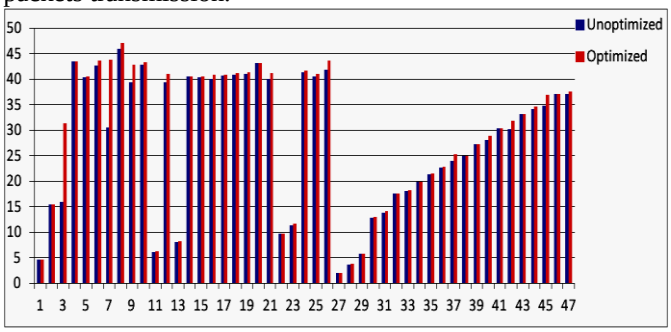


**Fig (1) Throughput Comparisons between Unoptimized and Optimized Network Cards**

**Table (1) The Throughputs of Unoptimized and Optimized Network Cards**

| Node ID | Transmission Size (GB) | Packet Size (Byte) | Default Setting Throughput (Mbps) | Optimal Setting Throughput (Mbps) | Improvement Rate (%) |
|---|---|---|---|---|---|
| 1 | 7.5 | 1400 | 4.583 | 4.584 | 0.02 |
| 2 | 6.9 | 536 | 15.40 | 15.45 | 0.32 |
| 3 | 9.9 | 576 | 15.86 | 31.39 | 97.92 |
| 4 | 21 | 1500 | 43.43 | 43.48 | 0.12 |
| 5 | 86 | 1400 | 40.27 | 40.44 | 0.42 |
| 6 | 19 | 1480 | 42.59 | 43.61 | 2.39 |
| 7 | 33 | 1460 | 30.58 | 43.76 | 43.10 |
| 8 | 61 | 1600 | 45.85 | 47.03 | 2.57 |
| 9 | 61 | 1440 | 39.37 | 42.74 | 8.56 |
| 10 | 14 | 1500 | 42.84 | 43.22 | 0.89 |
| 11 | 62 | 220 | 6.09 | 6.28 | 3.12 |
| 12 | 80 | 1400 | 39.31 | 40.97 | 4.22 |
| 13 | 61 | 280 | 8.02 | 8.28 | 3.24 |
| 14 | 62 | 1400 | 40.50 | 40.51 | 0.02 |
| 15 | 26 | 1400 | 40.41 | 40.59 | 0.45 |
| 16 | 21 | 1400 | 40.10 | 40.90 | 2 |
| 17 | 65 | 1400 | 40.60 | 40.90 | 0.74 |
| 18 | 16 | 1400 | 40.80 | 41.14 | 0.83 |
| 19 | 19 | 1400 | 41.04 | 41.27 | 0.56 |
| 20 | 7.9 | 1500 | 43.12 | 43.18 | 0.14 |
| 21 | 18 | 1400 | 40.10 | 41.15 | 2.62 |
| 22 | 21 | 340 | 9.69 | 9.72 | 0.31 |
| 23 | 7.4 | 400 | 11.40 | 11.61 | 1.84 |
| 24 | 17 | 1400 | 41.36 | 41.66 | 0.73 |
| 25 | 400 | 1400 | 40.44 | 41.04 | 1.48 |
| 26 | 368.64 | 1500 | 41.85 | 43.59 | 4.16 |

| | | | | | |
|---|---|---|---|---|---|
| 27 | 1 | 68 | 1.946 | 1.951 | 0.26 |
| 28 | 2 | 128 | 3.65 | 3.74 | 2.47 |
| 29 | 3 | 200 | 5.67 | 5.69 | 0.35 |
| 30 | 4 | 440 | 12.81 | 13 | 1.48 |
| 31 | 5 | 480 | 13.77 | 14.07 | 2.18 |
| 32 | 6 | 600 | 17.53 | 17.56 | 0.17 |
| 33 | 7 | 628 | 18.05 | 18.29 | 1.33 |
| 34 | 9 | 680 | 19.77 | 19.88 | 0.56 |
| 35 | 1.5 | 740 | 21.40 | 21.53 | 0.61 |
| 36 | 2.5 | 780 | 22.71 | 22.73 | 0.1 |
| 37 | 3.5 | 840 | 23.94 | 25.30 | 5.68 |
| 38 | 4.5 | 880 | 24.87 | 24.96 | 0.36 |
| 39 | 5.5 | 940 | 27.29 | 27.30 | 0.04 |
| 40 | 6.5 | 980 | 28.12 | 28.80 | 2.42 |
| 41 | 7.6 | 1040 | 30.27 | 30.39 | 0.40 |
| 42 | 9.5 | 1080 | 30.21 | 31.82 | 5.33 |
| 43 | 10 | 1140 | 33.08 | 33.13 | 0.15 |
| 44 | 10.5 | 1180 | 34.14 | 34.64 | 1.46 |
| 45 | 11 | 1240 | 34.74 | 36.83 | 6.02 |
| 46 | 11.5 | 1280 | 37.03 | 37.12 | 0.24 |
| 47 | 12 | 1340 | 37.13 | 37.50 | 1 |

The crash was a consequence of the CPU could not keep up generating and reacting to kernel interrupts as the stream of packets came. Subsequently, the crash resulted in approximately half of total packets lost. On the other hand, the optimized version managed to receive the transmission without crashing. It shows the high importance of kernel interrupts minimization to reduce CPU workload. This kernel interrupts minimization was achieved along with the maximization of throughput by the optimized network card. The values of the optimal network card settings for each transmission above can be seen in our previous paper [24]. The importance of kernel interrupt minimization is further affirmed by [22], which describes that kernel interrupt generation is actually an expensive process, because it requires a series of hardware and software events.

Special for the cases, where the optimal network card setting is active wait, instead of using the native active wait mode inside the PF_RING module in our real network transmissions, we implemented passive wait with 10ms, because in our mathematical model for active wait, 10ms of poll duration is chosen to represent the work of active wait [27].

Further, in general, regardless of transmission level, either small transmissions (less than 20GB), medium transmissions (between 20GB and 100GB inclusive), or big transmissions (above 100GB), the optimized versions of network cards consistently generate higher throughputs and lower CPU cycle consumption. It infers that our methodology to optimize network cards is applicable to various transmission levels.

## 6. CONCLUSION

In conclusion, optimization of network card setting increases throughput rate, and most importantly reduces CPU cycles utilization, thus protecting the CPU from being exhausted, over-heating, and crashing. Well maintained CPU will finally last longer, which brings economy value to the data centre. Finally, this conclusion proves that our methodology to optimize multiple network cards simultaneously, through simulation as published in [1], is workable and successful in real hardware environment. Added to the fact that our average simulation time between 2-3 hours, our methodology is considered to be practical as well.

## 7. ACKNOWLEDGEMENT

The authors are grateful for the support of the Ministry of Higher Education (MoHE), Malaysia (Grant No: 0153AB-K47) and Universiti Teknologi PETRONAS High Performance Cloud Computing Centre ($HPC^3$) for accommodating and funding this research.

## 8. REFERENCES

[1] Nurika, O., Hassan, M. F., Zakaria, N., Jung, L. T., "Genetic Algorithm Optimized Network in Cloud Data Centre" *Advance Science Letters Journal*, **22(10)**: 2705-2709 (2015).

[2] Nurika, O., Hassan, M. F., Zakaria, N., Jung, L. T., "Review of Cloud Network Optimization Practices" *Science International*, Lahore, **26(5)**: 1801-1805 (2014).

[3] Nurika, O., Hassan, M. F., Zakaria, N., Jung, L. T., "Workability Review of Genetic Algorithm Approach in Networks" *Proc. 2014 International Conference on Computer and Information Sciences (ICCOINS2014)*, 1-6 (2014).

[4] Scott, K., "On Proebsting's Law" *Technical Report*, University of Virginia, (2001).

[5] Sarraf, S., Brooks, J., Cole, J., Wang, X., "What Is The Impact of Smartphone Optimization on Long Surveys?" *2015 American Association for Public Opinion Research*, (2015).

[6] Holzle, U., Agesen, O., "Dynamic vs. Static Optimization Techniques for Object-Oriented Languages" *Journal Theory and Practice of Object Systems*, **1(3)**: 167-188 (1995).

[7] Menth, M., Martin, R., Hartmann, M., Sporlein, U., "Efficiency of Routing and Resilience Mechanisms in Packet-Switched Networks" *European Transactions on Telecommunications*, volume 21, **2(2)** (2009).

[8] Fritts, J. E., Steiling, F. W., Tucek, J. A., Wolf, W., "MediaBench II Video: Expediting the next generation of video systems research" *Microprocessors and Microsystems*, **33(4)**: 301-318 (2009).

[9] Fletcher, B. H., "FPGA Embedded Processors: Revealing True System Performance" *Embedded Systems Conference San Francisco 2005*, (2005).

[10] Sitharam, M., Peters, J., Zhou, Y., "Optimized parametrization of systems of incidences between rigid bodies" *Journal of Symbolic Computation*, **45(4)**: 481-498 (2010).

[11] Bai, Y. W., Cheng, C. H., "Dynamic Adjustment of CPU Clock Speed to Prevent Notebook Overheating

and Shutdown by AC Adapter," *The 1st IEEE Global Conference on Consumer Electronics 2012*, (2012).

[12] Best, B. N. B., "The Effect of Heat on Processors: A Challenge to Computer Usage in Northern Nigeria Rural Colleges" *Journal of Global Research in Computer Sciences*, (2015).

[13] Leitao, B. H., "Tuning 10Gb network cards on Linux," *Proceedings of the 2009 Linux Symposium*, (2009).

[14] Technologies, M., "Performance Tuning Guidelines for Mellanox Network Adapters (Revision 1.17 ed.)," *http://www.mellanox.com/related-docs/prod_software/Performance_Tuning_Guide_for_Mellanox_Network_Adapters.pdf*, (2016).

[15] Greer, C., Bob, A., Sammeta, S., "Maximizing File Transfer Performance Using 10Gb Ethernet and Virtualization," *http://www.intel.com/content/dam/support/us/en/documents/network/sb/fedexcasestudyfinal.pdf*, (2010).

[16] Emmerich, P., Raumer, D., Beifu, A., Erlacher, L., Wohlfart, F., et al., "Optimizing latency and CPU load in packet processing systems," *Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, Chicago, Illinois, (2015).

[17] Chang, X., Muppala, J. K., Zou, P., Li, X., "A Robust Device Hybrid Scheme to Improve System Performance in Gigabit Ethernet Networks," *32nd IEEE Conference on Local Computer Networks (LCN 2007)*, pp. 444-454, (2007).

[18] Maquelin, O., Gao, G. R., Hum, H. H., Theobald, K. B., Tian, X.-M., "Polling watchdog: Combining polling and interrupts for efficient message handling," *ACM SIGARCH Computer Architecture News*, pp. 179-188, (1996).

[19] Spertus, E., Dally, W. J., "Evaluating the locality benefits of active messages," *ACM SIGPLAN Notices*, pp. 189-198, (1995).

[20] Spertus, E., Goldstein, S. C., Schauser, K. E., Von Eicken, T., Culler, D. E., Dally, W. J., "Evaluation of mechanisms for fine-grained parallel programs in the J-machine and the CM-5," *ACM SIGARCH Computer Architecture News*, pp. 302-313, (1993).

[21] Dovrolis, C., Thayer, B., Ramanathan, P., "HIP: hybrid interrupt-polling for the network interface," *ACM SIGOPS Operating Systems Review*, **35**, pp. 50-60, (2001).

[22] Vahalia, U., "Unix Internals: The New Frontiers" *Dorling Kindersley Pvt. Limited*, (2008).

[23] Salah, K., Qahtan, A., "Implementation and experimental performance evaluation of a hybrid interrupt-handling scheme," *Computer Communications*, **32**, pp. 179-188, (2009).

[24] Nurika, O., Hassan, M. F., Zakaria, N., "A Study of Relations between Optimal Network Card Mode and Data Transmission Size" *Journal of Informatics and Mathematical Sciences*, (2016), In Press.

[25] hping3, *http://www.hping.org/hping3.html*.

[26] NTOP PF_RING, *http://www.ntop.org/products/packet-capture/pf_ring/*.

[27] Nurika, O., Hassan, M. F., Zakaria, N., Jung, L. T., "Mathematical Models for Network Card Simulation and Their Empirical Validations" *Proc. 2015 International Symposium on Mathematical Sciences & Computing Research (iSMSC2015)*, 66-71 (2015).